A Model Driven Approach to Agent-Based Service-Oriented Architectures

Ingo Zinnikus[1], Gorka Benguria[2], Brian Elvesæter[3], Klaus Fischer[1], and Julien Vayssière[4]

[1] DFKI GmbH, Stuhlsatzenhausweg 3 (Bau 43), D-66123 Saarbruecken, Germany.
[2] European Software Institute (ESI) - Corporacion Tecnologica Tecnalia - Parque Tecnológico de Zamudio, # 204 E-48170 Zamudio Bizkaia - Spain.
[3] SINTEF ICT, P.O. Box 124 Blindern, N-0314 Oslo, Norway
[4] SAP Research - Level 12 - 133 Mary Street - Brisbane QLD 4000 - Australia
Ingo.Zinnikus@dfki.de, Gorka.Benguria@esi.es, Brian.Elvesater@sintef.no, Klaus.Fischer@dfki.de, Julien.Vayssiere@sap.com

Abstract. Business process management has been identified as an interesting application area for agent technologies. Current developments in Web technologies support the execution of business processes in a networked environment. In this context, the flexible composition and usage of services in a service-oriented environment is a key feature. Additionally, the model-driven architecture (MDA) idea of transforming models on different abstraction levels, from highly abstract design-oriented views to an executable program, is a current trend in business process modeling. BDI agents provide a framework for both aspects by employing a planning from second principles approach, which uses a predefined library of plans and instantiates and adapts these plans. From this perspective, plans are design-time models for agent task execution and for Web Service composition. This paper presents a Rapid Prototyping framework for SOAs built around a Model-Driven Development methodology which we use for transforming high-level specifications of an SOA into executable artefacts, both for Web Services (WSDL files) and for BDI agents. The framework was designed to handle a mix of new and existing services and provides facilities for simulating, logging, analysing and debugging. Our framework was validated on a real industrial electronic procurement scenario in the furniture manufacturing industry. Once input from business experts had been collected, creating the high-level PIM4SOA (Platform Independent Model for SOA) model, deriving the Web service description and incorporating existing Web services took less than a day for a person already familiar with the techniques and tools involved. We show that rapid prototyping of SOAs is possible without sacrificing the alignment of the prototype with high-level architectural constraints.

1 Introduction

Service-oriented architectures (SOAs) have the potential to increase significantly the interoperability of information and communications technology (ICT) applications. However, business applications ask for planned and customizable services, which basically

K. Fischer et al. (Eds.): MATES 2006, LNAI 4196, pp. 110–122, 2006.
© Springer-Verlag Berlin Heidelberg 2006

comes down to the requirement for a methodology to do service composition in a flexible and efficient manner. The flexible combination and usage of services in such a service-oriented environment is a key feature that we believe can be best supported by agent technologies. Regarding service composition for an individual agent, we have two extreme options. On the one hand, we can try to adopt a general purpose planner and try to map service descriptions to individual actions which are then composed by the planner. In a naive approach, this will most likely result in a linear sequence of actions, i.e. service calls, that make up the newly composed service. It is quite unlikely that a general purpose planner will come-up with more complex structures. Business Process Modellers are often reluctant when confronted with the idea of choosing services or even processes at run-time. Control of the actual services invoked and the concrete processes performed is preferred over the possibility of an unperceived program decision. On the other hand, we can of course program the composition of services directly in some object-oriented programming language. In this case we are not restricted regarding the structures which we want to use. However, almost every change to a system which adopts such an approach is likely to end-up in painstaking programming sessions.

Composition languages such as e.g. BPEL4WS [1] address some of the problems arising from this approach, but still have limitations regarding flexibility and adaptability (especially during run-time). BPEL4WS does not *prevent* ways for choosing services at run-time, but a service endpoint discovered at run-time must adhere to e.g. a declared partner link and port type. Fault handlers allow the compensation of failures, but the concrete action to be performed in case of a failure has to be modeled in advance.

A planning from second principles approach which uses a predefined library of plans and instantiates and adapts these plans for the task at hand, when the system is at work, seems to bring together the best of two extreme approaches just described. The plan library can be maintained and incrementally updated and the agent will automatically take advantage of the knowledge which the plan library provides. How complex the plan structures in the plan library can be depends on the language which we use for specifying these plans.

SOAs as an architectural style for distributed systems have steadily been gaining momentum over the last few years and are now considered as mainstream in enterprise computing. Compared to earlier middleware products, SOAs put a stronger emphasis on loose coupling between the participating entities in a distributed system. The four fundamental tenets of Service Orientation [2] capture the essence of SOAs: explicit boundaries, autonomy of services, declarative interfaces, data formats and policy-based service description.

Web Services are the technology that is most often used for implementing SOAs. Web Services are a standards-based stack of specifications that enable interoperable interactions between applications that use the Web as a technical foundation [3]. The emphasis on loose coupling also means that the same degree of independence can be found between the organisations that build the different parts of an SOA. The teams involved only have to agree on service descriptions and policies at the level of abstraction prescribed by the different Web Service standards.

Our approach relies on a model-based approach to SOA prototyping that allows us to take existing services into account at a fairly high level of abstraction while keeping the

development of new components aligned with existing ones at each step of the process, from early modelling all the way down to execution and monitoring.

Section 2 introduces the framework for rapid prototyping for SOA. Section 3 details the model-driven development framework. Section 4 details the service enactment and monitoring platform. Section 5 presents how an autonomous agents framework can be used for performing the tasks of composition, mediation and brokering between Web Services. Section 6 introduces a detailed example based on a real industry scenario. Section 7 concludes and proposes avenues for future work.

2 A Framework for Rapid Prototyping of Service-Oriented Architectures

The framework for Rapid Prototyping of SOAs presented here is composed of three parts: a modelling part, a service part and an autonomous agent part. The modelling part is concerned with applying Model-Driven Development (MDD) techniques and tools to the design of SOAs. It defines models and transformations that are specific to the concepts used for SOAs, such as Web Service descriptions and plans for autonomous agents. The service part provides a highly flexible communication platform for Web services. The autonomous agent part deals both with designing and enacting service compositions as well as performing mediation, negotiation and brokering in SOAs.

Each of these three parts leverages the others in various ways. For example, the service part invokes the autonomous agents framework for starting the execution of a service composition described by a plan. The reverse also applies: autonomous agents may invoke Web Services through the tools from the services part. In turn, a description of a service composition at a platform-independent level can be transformed into a plan for autonomous agents, especially BDI agents (see Section 5). High-level service models can also be transformed into standards-based documents such as WSDL files.

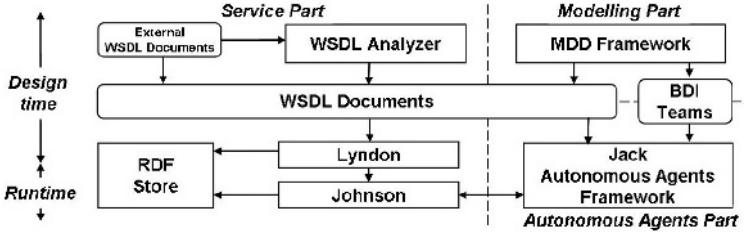

Fig. 1. High-level view of the SOA framework

Figure 1 gives a high-level overview of our framework, illustrating the main components as well as the flow of existing and generated artefacts such as WSDL files and BDI agent plans. The components of the framework are:

- The ***MDD framework*** defines the metamodels used to specify SOAs. It also provides modelling guidelines, model transformation and generation support for execution artefacts such as WSDL files and BDI plans. Importantly, it also supports importing existing WSDL files into the SOA models.
- The ***WSDL Analyzer*** is a tool for detecting similarities at a structural level between WSDL descriptions of Web services and generating the corresponding mappings. It supports flexible service invocation in a dynamic environment and the integration of external services.
- The ***Johnson*** tool is responsible for invoking Web services and receiving calls issued by Web service clients. The Lyndon tool takes WSDL files as input and configures Johnson for playing either the role of service provider, service consumer or service proxy for the service described by the WSDL file analyzed.
- The ***Jack*** [4] tool is used for specifying plans for autonomous agents which form teams that can invoke or receive calls from Web services. The plans used may be created as the result of an MDD process.
- The ***RDF store*** stores as RDF files both design-time information (WSDL files) and runtime information (SOAP messages) for the purpose of monitoring.

3 Model-Driven Development Framework for SOA

Designing SOAs at the enterprise level involves several different stakeholders within the enterprise. In order to support the various views pertinent to these stakeholders, we have defined an MDD framework. The MDD framework partitions the architecture of a system into several visual models at different abstraction levels subject to the concerns of the stakeholders. This allows important decisions regarding integration and interoperability to be made at the most appropriate level and by the best suited and knowledgeable people. The models are also subject for semi-automatic model transformations and code generation to alleviate the software development and integration processes.

Figure 2 details the model-driven development framework. It follows the OMG Model Driven Architecture (MDA) [5] approach and defines a Platform Independent Model (PIM) for SOA (PIM4SOA) and Platform Specific Models (PSMs) for describing Web services (XSD and WSDL), Jack BDI agents and BPEL [1] processes. The PIM4SOA is a visual PIM which specifies services in a technology independent manner. It represents an integrated view of the SOA in which different components can be deployed on different execution platforms. The PIM4SOA model helps us to align relevant aspects of enterprise and technical IT models, such as process, organisation and products models. This model allows us to raise the abstraction level at which we can talk about and reason on the architecture we design. The PIM4SOA metamodel defines modelling concepts that can be used to model four different aspects or views of a SOA:

1. **Service:** Services are an abstraction and an encapsulation of the functionality provided by an autonomous entity.
2. **Information:** Information is related to the messages or structures exchanged, processed and stored by software systems and components.
3. **Process:** Processes describe sequencing of work in terms of actions, control flows, information flows, interactions, protocols, etc.

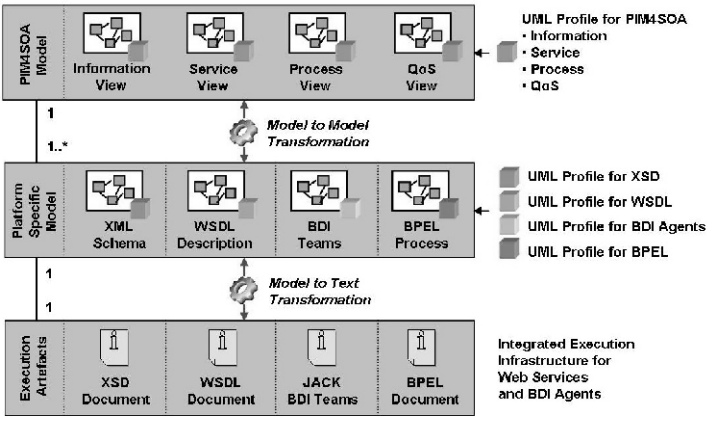

Fig. 2. Model-driven development framework

4. **Quality of service (QoS):** Extra-functional qualities that can be applied to services, information and processes.

The MDD framework provides model-to-model transformation services which allow us to transform PIM4SOA models into underlying PSMs such as XSD, WSDL, JACK BDI agents or BPEL. The PSMs depicted in Figure 2 are also visual models which IT developers can further refine by adding platform-specific modelling constructs such as deployment properties. PSMs typically represent a one-to-one mapping to an execution artefact. Dependencies between the various components are modelled at the PIM-level and two-way model transformations help us to ensure interoperability at the technical level and consistency at the PIM-level. Tool support for the MDD framework has been developed as a set of plugins for Rational Software Modeller (RSM) (IBM Rational Software). RSM is a UML 2.0 compliant modelling tool from IBM based on the Eclipse modelling environment. All models and metamodels were implemented using the EMF Core (Ecore) metamodel. Model transformations have been implemented using the model transformation capabilities of the RSM/Eclipse platform.

4 A Lightweight Web Services Enactment Framework

The part of our SOA Rapid Prototyping framework that deals with the enactment of Web services is composed of three tools which are arranged along a value chain: the WSDL Analyzer, the Lyndon tool and the Johnson tool.

4.1 The WSDL Analyzer

The WSDL Analyzer is a tool for detecting similarities between Web service descriptions. The tool can be used to find a list of similar services r to support the integration

of external services by producing a mapping between messages, thereby enabling brokering and mediation of services. The algorithm of the WSDL Analyzer improves over an algorithm for finding structural similarities proposed by Wang and Stroulia ([6], [7]) by taking into account additional features of the WSDL structure. More specifically, we make use of the tree-edit distance measure [8] and the concept of a weak subsumption relation introduced by Nagano et al. [9]. The idea of the tree-edit distance is that a similarity between two XML structures can be measured by stepwise transforming a tree representation of the first structure into the other. The steps necessary for that transformation provide the measure for their similarity, and, at the same time, induce the mapping between the schemas. Possible steps are basic edit operations such as node inserts, deletes and relabels. The algorithm of Wang and Stroulia considers only node matching without editing, or simple renaming operations such as changing a data type from string to int. Nagano et al. give three different types of weak subsumption: replacing labels, pruning edges and removing intermediate nodes. These operations can be correlated to specific tree-edit operations, namely relabeling and deleting nodes. A possible scenario for using the WSDL Analyzer is that the user already knows a service which provides the correct format. The WSDL of this service can be used as requirement for a similarity search. The WSDL Analyzer allows browsing the original WSDL and the candidate files. The algorithm detects common structures in port types, operations, messages and data type definitions. WordNet is integrated to improve the matching result. Mappings are assessed with a score which is used to establish a ranking between candidate service descriptions. The result is a ranking of the candidates according to their matching score. Based on the similarities, a mapping is generated between two WSDL descriptions which can be used to transform SOAP messages exchanged between similar services at runtime. The translation can be done automatically, if there is a one-to-one correspondence between elements. However, if there exist several possible corresponding elements, translation requires intervention from a user in order to unambiguously transform parameters. The latter case shows the limitation of the structural approach. There are possible mismatches which can be detected with the help of the WSDL Analyzer, but not automatically corrected.

4.2 The Johnson and Lyndon Tools

Johnson is a runtime tool that enables users to enact most of the roles typically found in an SOA, thereby enacting complex SOA scenarios by sending real SOAP messages between Web services without having to write a single line of code. Johnson features a Web-based user interface designed to closely resemble Web-based email applications, with the only difference that SOAP messages and Web Services endpoints are used in place of email messages and email addresses. The user can see incoming SOAP messages in the Inbox and create outgoing SOAP messages in the Outbox that will be sent to external Web services. A powerful user-interface generator relieves the user from having to deal with XML documents by generating forms for displaying and editing any XML-based data type. When playing the role of a Web service consumer, for example, a user would create a message in the Outbox, send that message to a remote Web service, and later see the response message appear in the Inbox. On the reverse, a user enacting a Web service provider would read incoming requests in the Inbox and reply to them

by creating response messages in the Outbox. Central to the architecture of Johnson are the concepts of endpoint, processing modules and processing chains. An endpoint is an abstraction for the address of a service. To each endpoint is attached a processing chain, which implements the processing of messages for that endpoint. Each processing chain is composed of a number of processing modules which are called in sequence. Each processing module is responsible for implementing one aspect of the processing of the message. A processing module may be responsible, for example, for adding routing information to the message, or dealing with reliability of the message exchange, or performing a transformation of the message content, or appending the message to an audit trail of messages sent. Assembling existing processing modules into new processing chains can be done through the user interface. Creating new processing modules, though, requires writing code.

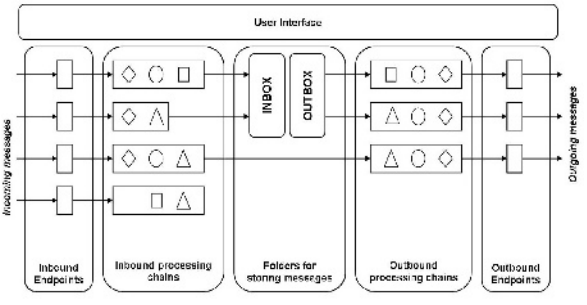

Fig. 3. Architecture of the Johnson tool

Figure 3 shows a sample instance of Johnson where all the aforementioned concepts are illustrated. The different processing modules are represented as different shapes such as circles, lozenges, squares and triangles. Messages received on the third inbound endpoint from the top do not end up in the inbox but are directly sent out using one of the outbound endpoints. This is possible because this logic is coded in the last processing module of the inbound processing chain. Being able to specialise message processing for each endpoint basis allows us to play the role of Web services that would implement different subsets of the Web Services stack of specifications, which proved very useful for studying the possible interoperability issues raised by the use of unrelated specifications together.

A processing module was also developed for keeping an audit trail of messages, which forms the basis for troubleshooting and performance measurement. The headers of SOAP messages are turned into RDF and stored in an RDF store. The Lyndon tool can be seen as the design-time counterpart of the Johnson tool. It analyses WSDL files and automatically configures Johnson for playing either the role of consumer or provider of the service described. Lyndon parses a WSDL file and determines which endpoints need to be created, and which processing chains need to be assigned to them. Determining which processing modules to include in the processing chain takes into account information extracted from the WSDL file as well as options set by the user.

The user may, for example, specify whether Johnson should be configured as a service consumer or a service provider, or whether messages sent to or from the service should be logged. Some configuration information can be extracted from the WSDL file, such as the need for implementing the WS-Addressing specification, which is specified as part of the description of the bindings of a Web service. Lyndon also generates an RDF representation of WSDL files and stores it into the same RDF store used for logging SOAP messages. Having both design-time and runtime artifacts in the same store is critical to monitoring the SOA and detecting services that do not behave accordingly to the service description they published.

5 An Agents-Based Execution Platform

The aim of the extended JACK agent framework for Web Services is to provide a goal-oriented service composition and execution module within an SOA. Following the Belief Desire Intention (BDI) model [10], agents are autonomous software components that have explicit goals to achieve or events to handle (desires). Agents are programmed with a set of plans to describe how to go about achieving desires. Each plan describes how to achieve a goal under varying circumstances. Set to work, the agent pursues its given goals (desires), adopting the appropriate plans (intentions) according to its current set of data (beliefs) about the state of the world. The combination of desires and beliefs initiating context-sensitive intended behaviour is part of what characterises a BDI agent. BDI agents exhibit reasoning behaviour under both pro-active (goal directed) and reactive (event driven) stimuli. Adaptive execution is introduced by flexible plan choice, in which the current situation in the environment is taken into account.

A BDI agent has the ability to react flexibly to failure in plan execution. Agents cooperate by forming teams in order to achieve a common goal. The JACK agent platform is not inherently ready for interaction within a Web service environment. Additional steps are necessary for enabling interactions between the agent platform and Web services, especially when the agents themselves offer services. In this case, some tools are needed for generating the server and client-side code for using JACK inside a Web server. Figure 4 is an overview of the extended JACK architecture for Web service composition and plan execution, with at its core the JACK agent framework with plan library and knowledge base. Following the MDA approach, a modeller specifies at design time a set of plans (PSM level) that constitute the workflow library of the agents. Web service calls are integrated as steps into plans. Workflows are modelled graphically and most of the common workflow patterns are supported.

In order to prepare the ground for a transformation from a PIM4SOA model to the JACK PSM, following the MDA approach, the service providers are mapped to Jack agents/teams. The parts of the PIM which define the processes involved are mapped to agent/team plans and correlated events, whereas the parts which define the interfaces are mapped to the modules which provide the client- and server-side code for the JACK agent platform. Hence, the idea is to apply a methodology which separates the Web service adapter from the core plans.

Accordingly, the steps of the methodology consist in

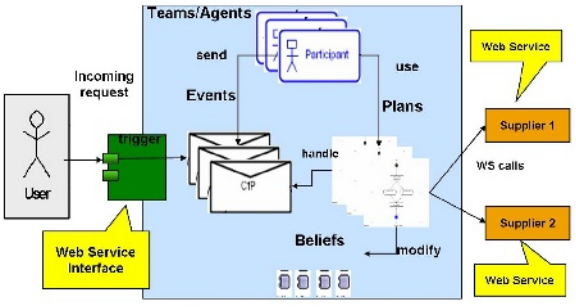

Fig. 4. Extended JACK framework for service composition and execution

- Constructing the agent/team plans for coordinating the internal activities and composing Web service calls based on the generated PSM models.
- Generate adapter code (client and server-side) with the help of a suitable toolkit (e.g. Apaches Axis framework [11]), based on a WSDL description of the service
- Adjust an agent's belief set so that it complies with the data structure provided by the Web services used and offered
- Specify events which correlate to Web service calls (in case the agents offer a Web service)
- Adjust the plans which handle these events
- Insert plan steps which trigger Web service calls (in case the agent calls a Web service) and call the adapter code from the plans directly

In order to compare an agent-based approach with other standards for Web service composition, the following distinction is useful [12]:

- Fixed composition: a fixed composite service requires the component services to be integrated in a fixed way. The composition structure and the component services are statically bound.
- Semi-fixed composition: in this situation, a composition definition which indicates the structure of the composition is generated. However, the actual service binding needs to be decided at run time.
- Explorative composition: this type of service composition is specified on the fly and requires dynamically structuring the composition service and binding component services.

Fixed composition can be done with e.g. BPEL4WS, but also by applying BDI agents. Semi-fixed composition might also be specified with BPEL4WS: partner links are defined at design time, but the actual service endpoint for a partner might be fixed at run-time, as long as the service complies with the structure defined at design time. Late binding can also be done with the JACK4WS framework. The service endpoint needs to be set (at the latest) when the actual call to the service is done. Explorative composition is beyond of what BPEL4WS and a BDI agent approach offer (at least if they are used in a 'normal' way). To enable explorative composition, a general purpose

planner might be applied which generates, based on the service descriptions stored in a registry, a plan which tries to achieve the objective specified by the consumer [13].

It might seem as if BPEL4WS and JACK4WS offer the same features. However, there are several advantages of a BDI agent approach which become evident by looking more closely at the definition of a semi-fixed composition. An important question is how the availability of a service is detected. This might be checked only by actually calling the service. If the service is not available or does not return the expected output, an exception will be raised. BPEL4WS provides a fault handler which allows specifying what to do in case of an exception. Similarly, a JACK agent plan will fail if a WS call raises an exception, and execute some activities specified for the failure case. However, the difference is that a plan is executed in a context which specifies conditions for plan instances and also other applicable plans. The context is implicitly given by the beliefs of an agent and can be made explicit. This means that in a given context, several plan instances might be executed, e.g. for all known services of a specific type, the services are called (one after another), until one of the services provides the desired result. An exception in one plan instance then leads to the execution of another plan instance for the next known service. Additionally, JACK provides the possibility of 'meta-level reasoning' which allows choosing the most feasible plan according to specified criteria. Similarly, if for a specific goal several plan types are feasible, the JACK execution engine executes one of these plans and, in case of a failure, immediately executes the next feasible plan to achieve the desired goal. The BDI agent approach supports this adaptive behaviour in a natural way, whereas a BPEL4WS process specification which attempts to provide the same behaviour would require awkward coding such as nested fault handlers etc. Another advantage is that *extending* the behaviour by adding a new, alternative plan for a specific task is straightforward. The new plan is simply added to the plan types and will be executed at the next opportunity. Similarly, *customizing* the composition is facilitated since the different plans clearly structure the alternatives of possible actions. Since the control structure is implicit, changes in a plan do not have impact on the control structure, reducing the danger of errors in the code.

6 An SOA Rapid Prototyping Case Study from the Furniture Industry

We have tested our approach against a real industry scenario, namely an electronic procurement process that spans the furniture manufacturing industry and the interior decoration retailers. The procurement process, traditionally, covers the activities that one organization performs to derive the goods to be purchased from the providers to build the products requested by the customer. We started by creating a PIM4SOA model based on the input we gathered from business experts at AIDIMA, a Spanish technology advisory body for the furniture industry. This PIM4SOA model details the interactions between the different roles involved in the e-procurement end-to-end process, from the initial customer order to the final acknowledgement of the delivery of the goods. To describe these interactions the PIM4SOA model identifies the different roles, the collaborations between those roles, the information exchanged, the internal behaviour of those collaborations and the expected quality that should be provided by the roles.

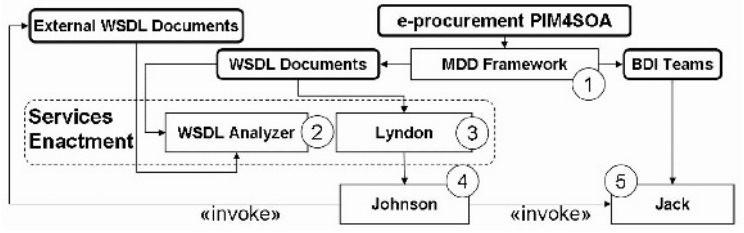

Fig. 5. Approach for the validation of the Rapid Prototyping framework

The following approach was followed for the validation of the Rapid Prototyping framework (see Figure 5). First we used the MDD framework (1) to derive the WSDL files and BDI models from the e-procurement PIM4SOA model. The next objective was to enact the services identified for the e-procurement scenario using the WSDL Analyser (2) and the Johnson and Lyndon (3) tool. Because some of the pieces of the SOA already existed, we used the WSDL Analyser to locate services similar to those required in the e-procurement scenario. For those that do not exist we have used the Lyndon tool to configure the Johnson platform to simulate them. The next step was to configure Johnson (4) to act as a service proxy; this allowed us to change the final service endpoints without affecting the process execution. Finally the PSM model for Jack (5) was implemented and tested with the enacted services.

- The ***MDD framework*** uses model-to-model transformations to derive the platform specific models for XML schemas, WSDL descriptions, and JACK Model from the PIM4SOA model as stated in the Section 3. These models are then completed by the platform experts to make them ready for the generation of the execution artefacts through the use of model-to-text transformations.
- The ***WSDL Analyzer*** compares the types of the parameters of the services required with the available services and returns a ranked set of candidate service. The technical experts then select the services that will be used and the tool provides the appropriate mappings to transform the messages at runtime.
- The ***Lyndon tool*** configures the Johnson tool for enacting some of the services and for logging all appropriate information in the RDF store for later analysing and debugging of the SOA.
- The ***Johnson tool*** is also configured to incorporate the endpoints of the mappings services generated by the WSDL Analyzer.
- Finally the ***Jack tool*** is loaded with the PSM-level model (agents/teams, plans, events, beliefs etc) for the e-procurement scenario.

Once we have implemented the prototype we can execute it together with the client to check that it achieves the stated requirements. If we need to analyse the details of the message exchange we can use the Johnson platform for doing so. Besides, the Johnson platform also allows us to simulate other situations in a flexible and agile way. Situations such as service delays, service shutdown or service errors can be simulated, logged and analysed.

7 Conclusion and Future Work

This paper presented a rapid prototyping framework for SOAs built around a Model-Driven Development methodology which is used for transforming high-level specifications of an SOA into executable artefacts, both in the realm of Web Services and in that of BDI agents. The framework can handle a mix of new and existing services and provides facilities for simulating, logging, analysing and debugging.

BDI agents, especially the JACK agents platform, provide the framework for a model-driven transformation of SOA specifications into executable processes and services. JACK agents execute the process descriptions and act as services within the SOA.

The framework was validated on a real industrial electronic procurement scenario from the furniture manufacturing industry. Once input from business expert had been collected, creating the high-level PIM4SOA model, deriving the Web service description and incorporating existing Web services took less than a day for a person already familiar with all the tools involved. After having run a few variants of the SOA, it became clear that the model-based approach we followed delivers significant value in keeping all the pieces of the SOA aligned with high-level business objectives throughout rounds of prototyping.

Acknowledgments

The work published in this paper is partly funded by the European Commission through the ATHENA IP (Advanced Technologies for interoperability of Heterogeneous Enterprise Networks and their Applications Integrated Project) (IST-507849). The work does not represent the view of the European Commission or the ATHENA consortium, and the authors are solely responsible for the papers content.

References

1. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business Process Execution Language for Web Services Version 1.1. Technical report (May 2003)
2. Box, D.: A guide to developing and running connected systems with indigo. MSDN Magazine (January 2004)
3. Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., Orchard, D.: Web Services Architecture. Technical report, W3C Working Group (February 2004)
4. The Agent Oriented Software Group: JACK Development Environment. http://www.agent-software.com (2004)
5. Soley, R.: Model Driven Architecture. OMG (November 2000)
6. Wang, Y., Stroulia, E.: Flexible Interface Matching for Web-Service Discovery. In: 4th Int'l Conf. on Web Information Systems Engineering (WISE 2003). (2003)
7. Wang, Y., Stroulia, E.: Semantic Structure Matching for Assessing Web-Service Similarity. In: 1st Intl Conf. on Service Oriented Computing (ICSOC 2003). Volume 2910 of Lecture Notes in Computer Science, Springer-Verlag (2003) pp. 197 – 207
8. Shasha, D., Zhang, K.: Approximate Tree Pattern Matching. In: Pattern Matching Algorithms. Oxford University Press (1997) 341 –371

9. Nagano, S., Hasegawa, T., Ohsuga, A., Honiden, S.: Dynamic Invocation Model of Web Services Using Subsumption Relations. In: IEEE International Conference on Web Services (ICWS). (2004) 150 – 156
10. Rao, A.S., Georgeff, M.P.: Modeling Rational Agents within a BDI-Architecture. In Allen, J., Fikes, R., Sandewall, E., eds.: Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91), Morgan Kaufmann publishers Inc.: San Mateo, CA, USA (1991) 473–484
11. Axis: (http://ws.apache.org/axis/)
12. Yang, J., Heuvel, W., Papazoglou, M.: Tackling the Challenges of Service Composition in e-Marketplaces. In: Proceedings of the 12th International Workshop on Research Issues on Data Engineering: Engineering E-Commerce/E-Business Systems (RIDE-2EC 2002), San Jose, CA, USA. (2002)
13. Sirin, E., Parsia, B., Wu, D., Hendler, J.A., Nau, D.S.: HTN planning for Web Service composition using SHOP2. J. Web Sem. **1** (2004) 377–396